PATENT ABSTRACTS OF JAPAN

(11)Publication number : 11-306038
(43)Date of publication of application : 05.11.1999

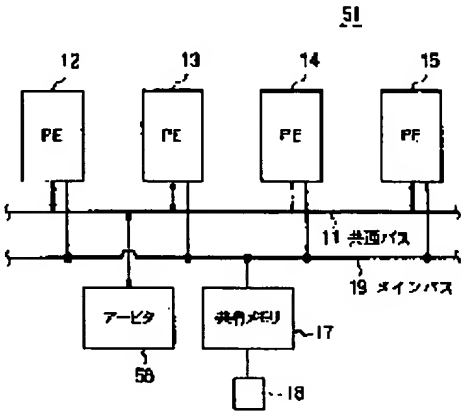
(51)Int.Cl. G06F 9/46
G06F 15/16

(21)Application number : 10-106756
(22)Date of filing : 16.04.1998

(71)Applicant : SONY CORP
(72)Inventor : IMAMURA YOSHIHIKO

(54) UNITS AND METHOD FOR PARALLEL OPERATION PROCESSING

(57)Abstract:
PROBLEM TO BE SOLVED: To provide a parallel operation units which can speedily and flexibly solve synchronization wait when plural tasks occur.
SOLUTION: A processor element PE 12 calls out plural tasks for PEs 13 to 15 and has a synchronization wait instruction performing synchronization wait as the necessity arises. An arbiter 56 increases a count value when task calling occurs and decreases the count value when the task is completed. The processor PE 12 compares the count value included in the synchronization wait instruction with the count value of the arbiter 56 when the synchronization waiting instruction is executed, releases the synchronization wait when the values coincide and performs the synchronization wait when they do not.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号

特開平11－306038

(43)公開日　平成11年(1999)11月5日

(51)Int.Cl.[6]	識別記号	ＦＩ	
Ｇ０６Ｆ　9/46	３６０	Ｇ０６Ｆ　9/46	３６０Ｆ
15/16	４３０	15/16	４３０Ｂ

審査請求　未請求　請求項の数11　ＯＬ　（全 10 頁）

(21)出願番号　特願平10－106756

(22)出願日　平成10年(1998)４月16日

(71)出願人　000002185
ソニー株式会社
東京都品川区北品川６丁目７番35号

(72)発明者　今村　義彦
東京都品川区北品川６丁目７番35号　ソニー株式会社内

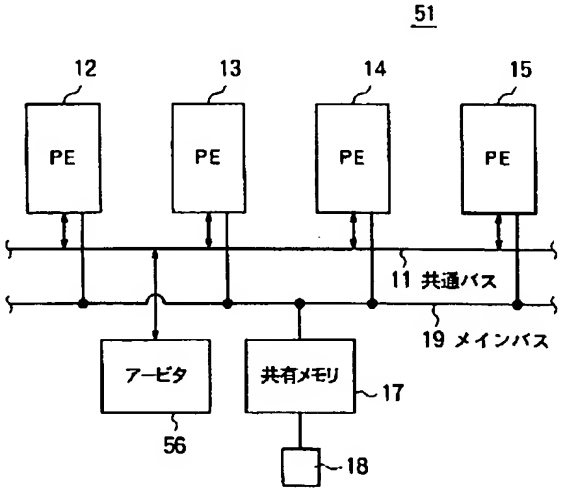
(74)代理人　弁理士　佐藤　隆久

(54)【発明の名称】　並列演算処理装置およびその方法

(57)【要約】

【課題】　複数のタスクを発生した場合の同期待ちを高速かつ柔軟に解決できる並列演算処理装置を提供する。

【解決手段】　プロセッサエレメントＰＥ１２がＰＥ１３～１５に対して複数のタスクを呼び出し、同期待ち命令で必要に応じて同期待ちを行う。アービタ５６は、タスク呼び出しが発生するとカウント値を増加し、そのタスクが終了するとカウント値を減少させる。プロセッサエレメントＰＥ１２は、同期待ち命令を実行したときに、同期待ち命令に含まれるカウント値と、アービタ５６のカウント値とを比較し、一致した場合に同期待ちを解除し、一致しない場合に同期待ちとなる。



1

【特許請求の範囲】

【請求項１】単数または複数のタスク呼び出し命令を実行した後に、同期待ち命令で必要に応じて同期待ちを行う第１の演算処理手段と、

前記第１の演算処理手段から呼び出されたタスクを実行し、当該呼び出されたタスクが終了したときにタスク終了命令を実行する単数または複数の第２の演算処理手段と、

前記第１の演算処理手段による前記タスク呼び出し命令の実行に応じてカウント値を増加し、前記第２の演算処理手段の前記タスク終了命令の実行に応じてカウント値を減少するカウント手段とを有し、

前記第１の演算処理手段は、前記同期待ち命令に含まれるカウント値と、前記カウント手段のカウント値とを比較し、当該比較の結果に応じて同期待ちを解除するか否かを決定する並列演算処理装置。

【請求項２】前記第１の演算処理手段は、前記同期待ち命令に含まれるカウント値と、前記カウント手段のカウント値とが一致したときに、同期待ちを解除する請求項１に記載の並列演算処理装置。

【請求項３】前記同期待ち命令に含まれるカウント値は、前記第１の演算処理手段から呼び出されているタスクの数に比べて小さい請求項２に記載の並列演算処理装置。

【請求項４】前記第１の演算処理手段における処理および前記単数または複数の第２の演算処理手段における処理とは相互に独立して行われる請求項１に記載の並列演算処理装置。

【請求項５】前記同期待ち命令は、前記カウント値を引数として持つ請求項１に記載の並列演算処理装置。

【請求項６】前記第１の演算処理手段および前記単数または複数の第２の演算処理手段は、共通のバスを介して接続されている請求項１に記載の並列演算処理装置。

【請求項７】第１の演算処理において、単数または複数のタスク呼び出し命令を実行し、

単数または複数の第２の演算処理において、前記第１の演算処理から呼び出されたタスクを実行し、当該呼び出されたタスクが終了したときにタスク終了命令を実行し、

前記第１の演算処理による前記タスク呼び出し命令の実行に応じて第１のカウント値を増加し、前記第２の演算処理の前記タスク終了命令の実行に応じて前記第１のカウント値を減少し、

前記第１の演算処理において、同期待ち命令が実行されたときに、前記同期待ち命令に含まれる第２のカウント値と、前記第１のカウント値とを比較し、当該比較の結果に応じて同期待ちを解除するか否かを決定する並列演算処理方法。

【請求項８】前記第１の演算処理は、前記同期待ち命令に含まれる第２のカウント値と、前記第１のカウント値

2

とが一致したときに、同期待ちを解除する請求項７に記載の並列演算処理方法。

【請求項９】前記同期待ち命令に含まれる第２のカウント値は、前記第１の演算処理から呼び出されているタスクの数に比べて小さい請求項８に記載の並列演算処理方法。

【請求項１０】前記第１の演算処理における処理および前記単数または複数の第２の演算処理における処理とは相互に独立して行われる請求項７に記載の並列演算処理方法。

【請求項１１】前記同期待ち命令は、前記カウント値を引数として持つ請求項７に記載の並列演算処理方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、複数のタスクを発生した場合の同期待ちを高速かつ柔軟に解決できる並列演算処理装置およびその方法に関する。

【０００２】

【従来の技術】例えば、複数のプロセッサエレメント（Processor Element:ＰＥ）が独立したプログラムカウンタを持ち、共通バスを介して相互に通信を行いながら処理を実行するＭＩＭＤ(Multiple Instruction Multiple Datastream)型のマルチプロセッサシステムが知られている。このようなマルチプロセッサシステムは、コンカレント（並行）マルチタスクを行うことを前提としており、タスクを生成しようとしているメインプログラムを実行しているプロセッサエレメントと、新たなタスクが生成されるプロセッサエレメントとの間で通信を行う。このとき、タスクを呼んだ（生成した）プログラムが、呼ばれた（生成された）タスクが終了するまで同期待ちを行う場合がある。

【０００３】図９は、一般的なマルチプロセッサシステム１の全体構成図である。図９に示すように、マルチプロセッサシステム１は、４つのプロセッサエレメントＰＥ１２，ＰＥ１３，ＰＥ１４，ＰＥ１５と、タスクの同期を管理するアービタ１６とが共通バス１１を介して接続されている。共通バス１１は、プロセッサエレメントＰＥ１２～１５の相互間でコマンドなどの制御信号を送受信するための制御線として機能する。

【０００４】また、マルチプロセッサシステム１では、プロセッサエレメントＰＥ１２，ＰＥ１３，ＰＥ１４，ＰＥ１５と共有メモリ１７とがメインバス１９を介して接続されている。共有メモリ１７は、外部端子１８を介して、外部メモリ（メインメモリ）に接続されている。なお、マルチタスクの同期を実現するマルチプロセッサシステムの構成としては、図９に示す構成以外にも種々のものがある。例えば、図９に示す例では、アービタ１６でタスクの同期が集中管理される場合を示したが、例えば、アービタ１６を設けずに、個々のプロセッサエレメントＰＥ１２～１５にタスクの同期を管理する機能を

持たせてもよい。

【０００５】図１０は、タスクを生成するプログラム（メインプログラム２５）が同期待ちを行う手法を説明するための図である。図１０に示す例では、プロセッサエレメントＰＥ１２上で動作するメインプログラム２５が、プロセッサエレメントＰＥ１３上にタスク２６を生成している。プロセッサエレメントＰＥ１２および１３は、個々のプロセッサ固有のマシン語（機械語）で記述された命令を実行して動作する。なお、タスクの生成ならびに同期の解決は、ハードウエアによる順序回路を用いても可能である。なお、本明細書では、命令によって同期機能の実現する場合を主に説明する。

【０００６】

【発明が解決しようとする課題】ところで、従来のマルチプロセッサシステムでは、以下に示す理由により、図１０に示すメインプログラム２５から複数のタスクを任意の数だけ生成することが困難であった。すなわち、マルチプロセッサシステムは、コンカレントマルチタスクを実行するが、このマルチタスク方式では、複数のプログラム（タスク）を複数のプロセッサエレメントＰＥに割り当てる必要がある。ここで、シングルプロセッサを前提としたマルチタスクでは、ＴＳＳ（Time Sharing System）方式のように、一つのプロセッサエレメントＰＥに対して時分割で複数のタスクを割り当てることが最も一般的である。従って、一つのプロセッサエレメントＰＥを中心としてタスク管理テーブルを一組だけ用意すればよい。このＴＳＳ方式を採用する多くの場合、それらタスクの切り替え機構を持つＵｎｉｘ（商標名）などのＯＳ(Operating System)を用いることになる。

【０００７】通常、プロセッサエレメントＰＥは、マルチタスクを特に意識した同期命令を用意していないことが多い。同期命令を用いるよりも、タイマなどの外部割り込みイベントを通じて例外処理を発生させ、その結果、タスクの切り替えを行う方式を採用することが多い。また、タスクの切り替えをより高速に実行するために、プロセッサエレメントＰＥ内でハードウエアによるサポートを行っていることが多いが、基本的にはソフトウエアによってタスク切り替え機能が実現される。

【０００８】これに対して、マルチプロセッサシステムでは、前述したＴＳＳ方式を採用する場合、複数のタスク管理テーブルを用意することが必要になる。また、それら複数のタスク管理テーブルを総合的に管理するプログラムを、個々のプロセッサエレメントＰＥを管理するプログラムの一階層上に用意する必要があり、かなり複雑なＯＳになる。そのため、従来のマルチプロセッサシステムでは、図１０に示すメインプログラム２５から複数のタスクを任意の数だけ生成することが困難であった。なお、マルチプロセッサシステムに実装するＯＳは、通常、そのマルチプロセッサシステムを使うサイド（ユーザ）が決定する。

【０００９】ところで、マルチタスクの実現方法は、ＴＳＳ方式の他にも存在する。例えば、特定のプロセッサエレメントＰＥをコプロセッサとして用いるなどのやや特殊用途への応用が考えられる。その他にも、特定のプロセッサエレメントＰＥをコプロセッサとして固定しないまでも、コプロセッサで実行するプログラムを常駐させることも、ある分野では非常に有効である。いずれにしても、マルチプロセッサシステムには、タスクの同期機構が必要である。

【００１０】研究試作段階のマルチプロセッサシステムでは、一般的に、個々のプロセッサエレメントＰＥ毎にシングルプロセッサと同じＯＳを搭載している。そして、これらプロセッサエレメントＰＥ間の通信を行うことにより、全体としてマルチタスクを達成していることが多い。この場合、プロセッサエレメントＰＥ相互の通信の機能の一部に同期機構が用いられる。あるいは、セマフォアなどのメモリを介した同期機構も採用可能である。

【００１１】しかしながら、タスクの生成とそのタスクの同期待ちに関しては、いずれの場合にも最終的にはソフトウエアによる処理を行っているのでレスポンスが悪く、せいぜい粗粒度並列プログラムを実行する場合に応用されるのが現状である。また、仮に、複数のタスクを生成することが十分可能なシステムであっても、それらのタスクの終了を待つ（同期待ち）方式に決定的な解決手段がない。

【００１２】メインプログラムから生成された複数のタスクのうちどのタスクを待つかなどの条件設定は、ソフトウエアによるプログラム記述によれば全ての組み合わせが可能であるが、それらの条件判定に費やす時間のオーバーヘッドもかなり大きくなり、高速に同期をとることができない。一方、このような設定条件を、ハードウエアによって決定するものがある。例えば、米国のインテル社が開発したマイクロプロセッサ８０８６とそのプロセッサ専用に設計されたコプロセッサ８０８７との間では、ハンドシェークによる同期待ちシステムが確立されている。プロセッサ８０８６上のメインプログラムの数値演算用の命令を実行すると、自動的にコプロセッサ８０８７はその命令を解釈して演算を始める。通常、数値演算の実行には複数のクロックサイクルが必要とされていた。従って、その間、プロセッサ８０８６では当該命令の次の命令を順次実行する。

【００１３】メインプログラムには、タスクの発生命令から適当な命令数を経た後に同期命令が記述される。そして、その同期命令が実行される前に当該数値演算が終了していれば、プロセッサ８０８６は数値演算が同期されているものとみなしそのまま命令実行を進める。また、その同期命令が実行される前に当該数値演算が終了していなければ、コプロセッサ８０８７の演算が終了するまで同期待ちをする。この同期待ちシステムは、簡単

なプロトコルによるハンドシェーク信号を用いており、きわめて簡単な構成で高速に同期をとることができる。しかしながら、複数のコプロセッサ８０８７をひとつのプロセッサ８０８６に接続することができず、複数のタスクを生成した場合の同期待ちを解決できないという問題がある。

【００１４】本発明は上述した従来技術に鑑みてなされ、複数のタスクを発生した場合の同期待ちを高速かつ柔軟に解決できる並列演算処理装置およびその方法を提供することを目的とする。

【００１５】

【課題を解決するための手段】本発明は、前述した従来の同期待ち機構に改良を加えて、マルチタスクシステムの全体的な性能を向上させるものである。先に記述したマルチプロセッサシステムにおける同期機構において、メインプログラムから生成されたタスク（以下、子タスクとも記す）の終了を認識する際に要する時間が長いという問題があったが、これは、従来のマルチプロセッサシステムが汎用性を重視するあまり、ソフトウエアによる解決方法を選択したことに原因がある。

【００１６】本発明は、タスクの生成とその同期に伴う汎用性をある程度制限している。タスクの生成はほぼ自由に任意の数分を許すものとし、それら子タスクの稼動状況／終了を自動的に数値（カウント値）に換算する。メインプログラムでは、同期命令の実行条件（成立条件）にその数値を含める。そして、子タスクの数値化された稼動状況と同期命令を組み合わせて用いる。これは、複数のタスクの生成を可能とする。また、同期命令は、子タスクの終了をハードウエアによって認識させる方法を採用し、レスポンスの高速化を図る。

【００１７】ここで、本発明を実現する手段として、子タスクを生成する毎にメインプログラムを実行しているプロセッサエレメントは、生成した子タスクの数を記憶しておくことを提案する。これは、単にレジスタと加減算器とを組み合わせてもよいし、カウンタを用意するだけでもよい。カウンタを例にとると、初期値として０をセットしておき、子タスクを生成する毎に１だけインクリメント（増加）させていく。そして、子タスクが終了すると、しかるべき手続きを経てメインプログラムを実行しているプロセッサエレメントＰＥにその旨を通知して、先のカウント値を１だけ減ずる。

【００１８】メインプログラムのなかの同期命令を実行するプロセッサエレメントは、同期命令の実行時に前述したカウント値とその同期命令に付加された引数の値とを比較する。比較の結果、カウント値のほうが小さいかあるいは同じ値であれば同期条件が成立したものとみなし、メインプログラムはその同期命令以降の命令実行に進む。もし、そうでなければ（カウント値のほうが大きければ）、同期条件を満たすまで同期待ちする。

【００１９】同期命令実行後は、次に子タスクの生成に

備えて０に初期化をすることにしてもよい。また、そうしなくてもよい。これはそのシステムを使用する利用者の課題である。また、カウンタを設ける位置は特に限定されない。例えば、メインプログラムを実行するプロセッサエレメント内に存在させるか、あるいは、アービタモジュール内に存在させてもよい。

【００２０】従来の同期機構では、仮に複数の子タスクを生成したときには、ソフトウエアによりすべての詳細な同期条件を記述するか、あるいは、ハードウエアによりすべての子タスクの終了を待つかの方法を採用していた。これに対して、本発明は、子タスクの終了をハードウエアによって認識させるわけであるが、同期待ち命令に伴う同期条件に制限を付加して、単に同期させるべき子タスクの数のみを設定する。これにより、同期機構にある程度の柔軟性を持たせながら論理回路の規模削減を図ることができる。

【００２１】すなわち、本発明の並列演算処理装置は、単数または複数のタスク呼び出し命令を実行した後に、同期待ち命令で必要に応じて同期待ちを行う第１の演算処理手段と、前記第１の演算処理手段から呼び出されたタスクを実行し、当該呼び出されたタスクが終了したときにタスク終了命令を実行する単数または複数の第２の演算処理手段と、前記第１の演算処理手段による前記タスク呼び出し命令の実行に応じてカウント値を増加し、前記第２の演算処理手段の前記タスク終了命令の実行に応じてカウント値を減少するカウント手段とを有し、前記第１の演算処理手段は、前記同期待ち命令に含まれるカウント値と、前記カウント手段のカウント値とを比較し、当該比較の結果に応じて同期待ちを解除するか否かを決定する。

【００２２】また、本発明の並列演算処理装置は、好ましくは、前記第１の演算処理手段は、前記同期待ち命令に含まれるカウント値と、前記カウント手段のカウント値とが一致したときに、同期待ちを解除する。

【００２３】また、本発明の並列演算処理装置は、好ましくは、前記同期待ち命令に含まれるカウント値は、前記第１の演算処理手段から呼び出されているタスクの数に比べて小さい。

【００２４】さらに、本発明の並列演算処理方法は、第１の演算処理において、単数または複数のタスク呼び出し命令を実行し、単数または複数の第２の演算処理において、前記第１の演算処理から呼び出されたタスクを実行し、当該呼び出されたタスクが終了したときにタスク終了命令を実行し、前記第１の演算処理による前記タスク呼び出し命令の実行に応じて第１のカウント値を増加し、前記第２の演算処理の前記タスク終了命令の実行に応じて前記第１のカウント値を減少し、前記第１の演算処理において、同期待ち命令が実行されたときに、前記同期待ち命令に含まれる第２のカウント値と、前記第１のカウント値とを比較し、当該比較の結果に応じて同期

待ちを解除するか否かを決定する。

【００２５】

【発明の実施の形態】以下、本発明の実施形態に係わるマルチプロセッサシステムを説明する。図１は、本実施形態のマルチプロセッサシステム５１の構成図である。図１に示すように、マルチプロセッサシステム５１は、４つのプロセッサエレメントＰＥ１２，ＰＥ１３，ＰＥ１４，ＰＥ１５と、タスクの同期を管理するアービタ５６とが共通バス１１を介して接続されている。また、マルチプロセッサシステム５１では、プロセッサエレメントＰＥ１２，ＰＥ１３，ＰＥ１４，ＰＥ１５と共有メモリ１７とがメインバス１９を介して接続されている。共有メモリ１７は、外部端子１８を介して、外部メモリ（メインメモリ）に接続されている。

【００２６】図１において、図９と同じ符号を付したプロセッサエレメントＰＥ１２，ＰＥ１３，ＰＥ１４，ＰＥ１５、共通バス１１、メインバス１９、共有メモリ１７および外部端子１８は、前述した図９に示す構成要素と同じである。すなわち、マルチプロセッサシステム５１は、アービタ５６におけるタスク同期の管理方法に特徴を有する。アービタ５６は、カウンタを備え、例えば、プロセッサエレメントＰＥ１２が子タスクの生成を伴う命令を実行したときに、カウンタのカウント値を１だけ増加し、子タスクが終了したときにカウントのカウント値を１だけ減少させる。

【００２７】マルチプロセッサシステム５１は、プロセッサエレメントＰＥ１２～１５が、他のプロセッサエレメントＰＥに対して任意に子タスクを生成することができる。子タスクはそのプログラムの終了時に、当該子タスクを呼び出した親のプログラムに対して、子タスクの終了を示すメッセージを送信する。このメッセージは、図１の共通バス１１を用いて行われるものとするが、その形態はいかなるものであってもよい。プロセッサ間通信を厳密に規定してもよいし、単に信号線を通じて知らせてもよい。

【００２８】なお、説明の都合上、子タスクとして生成されるプログラムのアドレス（メインメモリ上の番地）は予めメインプログラムの記述のなかで設定されているものとする。また、子タスクは、プロセッサエレメントＰＥ１２以外のプロセッサエレメントＰＥ１３～１５上に生成されるものとする。具体的にどのプロセッサエレメントＰＥに生成するかは図１に示すアービタ５６が自動的に決定する。本実施形態では、子タスクの割り当て方法について触れない。

【００２９】子タスクを生成する命令としては、例えば、「ｇｅｎ」命令を用いるが、その名前は任意である。従って、「ｇｅｎ」命令がプリミティブなマシン語ではなく、アセンブラ言語のマクロ命令であっても良い。子タスクのプログラムの中で、タスク生成やその同期、終了に関するもの以外の通常の命令を「ｉｎｓｔ１」、「ｉｎｓｔ２」と記述する。これも前記と同様に、それら命令の名前や構成方法は任意である。また、子タスクの終了命令を「ｅｎｄ」とする。この命令によって、子タスクを呼んだプロセッサエレメントＰＥに対して、タスクの終了を示すメッセージを自動的に送る。子タスクで作成したデータその他の情報は、適当に処理するものとする。また、メインプログラムにおいて、同期待ちのための命令を「ｗａｉｔ」とする。後述するが、この「ｗａｉｔ」命令には引数を設定する事ができる。

【００３０】以下、マルチプロセッサシステム５１の動作を説明する。単数の子タスクのみを生成する場合図２はマルチプロセッサシステム５１において一つの子タスクを生成する場合のプロセッサエレメントＰＥ上で動作するプログラムを説明するための図、図３は図２に示す場合におけるタスク発生および同期待ち解除のタイミングを説明するための図である。ここで、メインプログラム２５の「ｗａｉｔ」命令は、アービタ５６のカウンタのカウント値が「０」になることを同期解除条件としている。例えば、図３に示すタイミング「ｒ１」で、プロセッサエレメントＰＥ１２において、図２に示すメインプログラム２５に含まれる「ｇｅｎ」命令が実行され、タイミング「ｎ１」でプロセッサエレメントＰＥ１３上にタスク２６が生成される。このとき、アービタ５６のカウント値が「１」に設定される。そして、プロセッサエレメントＰＥ１３において、タスク２６の「ｉｎｓｔ１」および「ｉｎｓｔ２」などの命令が実行される。また、プロセッサエレメントＰＥ１２において、メインプログラム２５の「ｇｅｎ」命令以降の命令が実行される。そして、図３に示すタイミング「ｓ１」で、プロセッサエレメントＰＥ１２において、図２に示す「ｗａｉｔ」命令が実行される。このとき、アービタ５６のカウント値が「１」であるため、プロセッサエレメントＰＥ１２は、同期条件が満たされていないと判断し、同期待ち状態になる。

【００３１】そして、図３に示すタイミング「ｓ２」で、プロセッサエレメントＰＥ１３においてタスク２６の「ｅｎｄ」命令が実行されると、タスク２６が終了すると共に、アービタ５６がカウント値が１だけ減算されて「０」になる。これにより、同期条件が成立し、プロセッサエレメントＰＥ１２は、メインプログラム２５の「ｗａｉｔ」命令以降の命令を実行する。なお、図２および図３に示す場合において、プロセッサエレメントＰＥ１２が、メインプログラム２５に含まれる「ｗａｉｔ」命令が実行される前に、プロセッサエレメントＰＥがタスク２６の「ｅｎｄ」命令を実行した場合には、プロセッサエレメントＰＥ１２は同期待ち状態にならない。

【００３２】複数の子タスクを生成する場合（その１）図４はマルチプロセッサシステム５１において複数のタ

スクを生成し、全てのタスクの終了を同期待ちの条件とした場合の各プロセッサエレメントＰＥ上で動作するプログラムを説明するための図、図５は図４に示す場合におけるタスク発生および同期待ち解除のタイミングを説明するための図である。図４および図５に示す例では、図４に示すメインプログラム７５の「ｗａｉｔ」命令が示す同期待ち解除条件がカウント値「０」となっている。この場合には、図５に示すタイミング「ｒ１」で、プロセッサエレメントＰＥ１２において、図４に示すメインプログラム７５に含まれる「ｇｅｎ１」命令が実行され、タイミング「ｒ２」でプロセッサエレメントＰＥ１３上にタスク７６が生成される。このとき、アービタ５６のカウント値が「１」に設定される。また、図５に示すタイミング「ｒ２」で、プロセッサエレメントＰＥ１２において、図４に示すメインプログラム７５に含まれる「ｇｅｎ２」命令が実行され、タイミング「ｒ３」でプロセッサエレメントＰＥ１４上にタスク７７が生成される。このとき、アービタ５６のカウント値が「２」に設定される。また、図５に示すタイミング「ｒ３」で、プロセッサエレメントＰＥ１２において、図４に示すメインプログラム７５に含まれる「ｇｅｎ３」命令が実行され、タイミング「ｎ１」でプロセッサエレメントＰＥ１５上にタスク７８が生成される。このとき、アービタ５６のカウント値が「３」に設定される。

【００３３】そして、プロセッサエレメントＰＥ１３において、タイミング「ｒ２」から、タスク７６に含まれる「ｉｎｓｔ１」および「ｉｎｓｔ２」などの命令が実行される。また、プロセッサエレメントＰＥ１４において、タイミング「ｒ３」から、タスク７７に含まれる「ｉｎｓｔ１」および「ｉｎｓｔ２」などの命令が実行される。また、プロセッサエレメントＰＥ１５において、タイミング「ｎ１」から、タスク７８に含まれる「ｉｎｓｔ１」および「ｉｎｓｔ２」などの命令が実行される。

【００３４】そして、図５に示すタイミング「ｅ１」で、プロセッサエレメントＰＥ１３においてタスク７６の「ｅｎｄ」命令が実行されると、タスク７６が終了すると共に、アービタ５６のカウント値が１だけ減算されて「２」になる次に、図５に示すタイミング「ｓ１」で、プロセッサエレメントＰＥ１２において、図４に示す「ｗａｉｔ」命令が実行される。このとき、アービタ５６のカウント値が「２」であることから、同期待ち解除条件を満たさず、プロセッサエレメントＰＥ１２は同期待ち状態になる。

【００３５】次に、図５に示すタイミング「ｅ２」で、プロセッサエレメントＰＥ１５においてタスク７８の「ｅｎｄ」命令が実行されると、タスク７８が終了すると共に、アービタ５６がカウント値が１だけ減算されて「１」になる次に、図５に示すタイミング「ｓ２」で、プロセッサエレメントＰＥ１４においてタスク７７の「ｅｎｄ」命令が実行されると、タスク７７が終了すると共に、アービタ５６がカウント値が１だけ減算されて「０」になるこれにより、メインプログラム７５の「ｗａｉｔ」命令が示す同期待ち解除条件が満たされ、プロセッサエレメントＰＥ１２の同期待ちが解除される。

【００３６】複数の子タスクを生成する場合（その２）

図６はマルチプロセッサシステム５１において複数のタスクを生成し、２つのタスクの終了を同期待ちの条件とした場合の各プロセッサエレメントＰＥ上で動作するプログラムを説明するための図、図７は図６に示す場合におけるタスク発生および同期待ち解除のタイミングを説明するための図である。図６および図７に示す例では、図６に示すメインプログラム８５の「ｗａｉｔ」命令が示す同期待ち解除条件がカウント値「１」となっている。この場合には、図７に示すタイミング「ｒ１」で、プロセッサエレメントＰＥ１２において、図６に示すメインプログラム８５に含まれる「ｇｅｎ１」命令が実行され、タイミング「ｒ２」でプロセッサエレメントＰＥ１３上にタスク８６が生成される。このとき、アービタ５６のカウント値が「１」に設定される。また、図７に示すタイミング「ｒ２」で、プロセッサエレメントＰＥ１２において、図６に示すメインプログラム８５に含まれる「ｇｅｎ２」命令が実行され、タイミング「ｒ３」でプロセッサエレメントＰＥ１４上にタスク８７が生成される。このとき、アービタ５６のカウント値が「２」に設定される。また、図７に示すタイミング「ｒ３」で、プロセッサエレメントＰＥ１２において、図６に示すメインプログラム８５に含まれる「ｇｅｎ３」命令が実行され、タイミング「ｎ１」でプロセッサエレメントＰＥ１５上にタスク８８が生成される。このとき、アービタ５６のカウント値が「３」に設定される。

【００３７】そして、プロセッサエレメントＰＥ１３において、タイミング「ｒ２」から、タスク８６に含まれる「ｉｎｓｔ１」および「ｉｎｓｔ２」などの命令が実行される。また、プロセッサエレメントＰＥ１４において、タイミング「ｒ３」から、タスク８７に含まれる「ｉｎｓｔ１」および「ｉｎｓｔ２」などの命令が実行される。また、プロセッサエレメントＰＥ１４において、タイミング「ｎ１」から、タスク８８に含まれる「ｉｎｓｔ１」および「ｉｎｓｔ２」などの命令が実行される。

【００３８】そして、図７に示すタイミング「ｅ１」で、プロセッサエレメントＰＥ１３においてタスク８６の「ｅｎｄ」命令が実行されると、タスク８６が終了すると共に、アービタ５６のカウント値が１だけ減算されて「２」になる次に、図７に示すタイミング「ｓ１」で、プロセッサエレメントＰＥ１２において、図６に示す「ｗａｉｔ」命令が実行される。このとき、アービタ５６のカウント値が「２」であることから、同期待ち解除条件を満たさず、プロセッサエレメントＰＥ１２は同

期待ち状態になる。

【００３９】次に、図５に示すタイミング「ｅ２」で、プロセッサエレメントＰＥ１５においてタスク８８の「ｅｎｄ」命令が実行されると、タスク８８が終了すると共に、アービタ５６がカウント値が１だけ減算されて「１」になる。これにより、メインプログラム７５の「ｗａｉｔ」命令が示す同期待ち解除条件が満たされ、プロセッサエレメントＰＥ１２の同期待ちが解除される。

【００４０】このように、図６および図７に示す例では、図６に示すメインプログラム８５が生成した３つの子タスク８６，８７，８８のうち２つの子タスクの終了を「ｗａｉｔ」命令の引数に記述する。その結果、タスク８６および８８が終了すると、タスク８７の終了を待たずに同期待ち状態が解除できる。ここで、タスク８７としては、例えば、仮想記憶をサポートするタスクなどの長期的に存在するタスクが適用される。

【００４１】以上説明したように、マルチプロセッサシステム５１によれば、マルチタスクを実現する場合でも、ＯＳなどによる複雑な管理を行うことなく、複数のタスクを発生したプログラムの同期待ちを解決することができる。マルチプロセッサシステム５１によるタスク管理は、ソフトウエアによるタスクの管理と比べてやや汎用性に制限が付くものの、同期待ち機構をハードウエアで実現することがきわめて容易になる。これは、タスク終了の事実を高速に知るための方法として有効であり、また、回路規模が小さく実現できるなどの効果がある。

【００４２】また、同期待ち命令である「ｗａｉｔ」命令に引数を記述することにより、メインプログラムと、当該メインプログラムが生成した子タスクとの間の同期条件を柔軟に設定できる。例えば、マルチプロセッサシステム５１は、図６において、プロセッサエレメントＰＥ１４上に生成される子タスク８７が他の子タスク８６，８８に比べてプログラムが終了するまでの時間が極めて大きくなることが事前に知ることができていて、なおかつ、プロセッサエレメントＰＥ１２上の同じメインプログラム８５で子タスクを生成する必要性にせまられたときなどは特に有効である。

【００４３】本実施形態では、意図的に極端な場合を想定している。しかし、タスクの処理時間に大小の差が存在することは一般的にいえることである。マルチプロセッサシステム上でマルチタスクを実現するためには、何らかの方法でプロセッサ資源と各々のスレッド（ＯＳ上の概念でタスクの断片のこと）とを対応づけすることが求められる。プロセッサ資源を有効に使用するためには、システム上のプロセッサ稼動状況を的確に把握する必要があり、また、ある特定のプロセッサの動作状況によってシステム全体が影響を受けることはあまり望ましくはない。従来の技術を用いてマルチプロセッサシステム上にマルチタスクを実現する場合には、前述したように、プロセッサ資源を有効に使用することが困難になる。つまり、ソフトウェアによる制御では稼動していないプロセッサに対して新たなタスクを割り当てることが可能であるが、その判断を下すまでに多くの時間を費やすことになる。結果的に、システム全体が遅く動作することになる。

【００４４】これに対して、マルチプロセッサシステム５１によれば、同期待ちの解決手段として、カウンタを用いるのみで、複雑なソフトウェアを用いたないため、同期待ちを高速に解決でき、リアルタイム性を向上できる。また、ハードウエアによる従来の技術では、子タスクの生成とその同期機構を高速にかつ簡単に構成することができる一方で、複数のタスクを生成することが困難になっている。仮に、複数のタスクが生成できたとしても、それまでに生成したタスクをすべて待つことになる。これに対して、マルチプロセッサシステム５１によれば、「ｗａｉｔ」命令に同期待ちを行うタスクの数を引数として設定することで、多様かつ柔軟な同期待ちを実現できる。

【００４５】ＯＳを実装する上で、新たなタスクを生成する際にはそれまで生成されているタスクのうちいくつまでが終了しているかなどを把握し、かつ、それらのタスクがどのプロセッサエレメントＰＥに割り当てられているかを知る必要がある。裏を返せば、すべてのタスクの終了を待っていては、新たなタスクのプロセッサエレメントＰＥへの割り当てができないことになるか、あるいは、相当の困難を伴うことになるだろう。従って、上述したマルチプロセッサシステム５１のように、タスクを生成するべきメインプログラム内でタスクの終了状態を監視しておきその値を同期機構に反映させることができれば、マルチプロセッサのシステムを向上させることができる。

【００４６】本発明は上述した実施形態には限定されない。上述した実施形態では、図１に示すプロセッサエレメントＰＥ１２がタスク生成命令である「ｇｅｎ」命令を実行する場合を例示したが、その他のプロセッサエレメントＰＥ１３～１５が「ｇｅｎ」命令を実行してもよい。また、図１に示す例では、４個のプロセッサエレメントＰＥ１２～１５を設けた場合を例示したが、２以上であればプロセッサエレメントＰＥの数は任意である。

【００４７】上述した実施形態では、アービタ５６にタスク同期機能を持たせた場合を例示したが、タスク同期機能をその他の構成要素に持たせてもよい。

【００４８】また、本発明は、例えば、図８に示すように、コンピュータ１０２，１０３，１０４，１０５をネットワーク１０１を介して接続して分散処理を行う並列分散処理システムにも適用できる。この場合に、例えば、コンピュータ１０２がメインプログラムを実行する場合には、コンピュータ１０２内にアービタ１０６が設

13

けられる。アービタ１０６の機能は、前述した図１に示すアービタ５６と同じである。

【００４９】

【発明の効果】以上説明したように、本発明の並列演算処理装置によれば、第１の演算処理手段が複数のタスクを呼び出した場合の同期待ちを高速かつ柔軟に解決できる。また、本発明の並列演算処理方法によれば、第１の演算処理が複数のタスクを呼び出した場合の同期待ちを高速かつ柔軟に解決できる。

【図面の簡単な説明】

【図１】図１は、本発明のマルチプロセッサシステムの構成図である。

【図２】図２は、図１に示すマルチプロセッサシステムにおいて一つの子タスクを生成する場合のプロセッサエレメントＰＥ上で動作するプログラムを説明するための図である。

【図３】図３は、図２に示す場合におけるタスク発生および同期待ち解除のタイミングを説明するための図である。

【図４】図４は、図１に示すマルチプロセッサシステムにおいて複数のタスクを生成し、全てのタスクの終了を同期待ちの条件とした場合の各プロセッサエレメントＰＥ上で動作するプログラムを説明するための図である。

14

【図５】図５は、図４に示す場合におけるタスク発生および同期待ち解除のタイミングを説明するための図である。

【図６】図６は、図１に示すマルチプロセッサシステムにおいて複数のタスクを生成し、２つのタスクの終了を同期待ちの条件とした場合の各プロセッサエレメントＰＥ上で動作するプログラムを説明するための図である。

【図７】図７は、図６に示す場合におけるタスク発生および同期待ち解除のタイミングを説明するための図である。

【図８】図８は、本発明を適用した並列分散処理システムの構成図である。

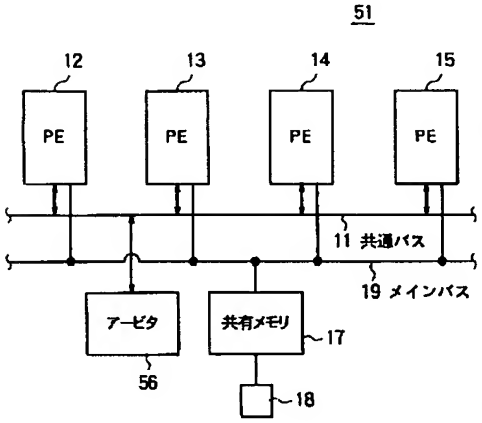
【図９】図９は、一般的なマルチプロセッサシステムの全体構成図である。

【図１０】図１０は、タスクを生成するプログラムが同期待ちを行う手法を説明するための図である。

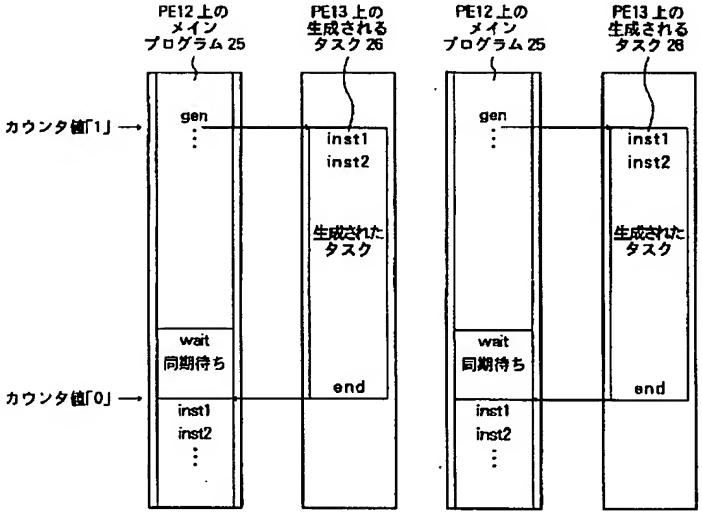
【符号の説明】

１，５１…マルチプロセッサシステム、１１…共通バス、１６，５６，１５６…アービタ、１７…共有メモリ、１９…メインバス、１２～１５…プロセッサエレメントＰＥ、１８…外部端子、１００…並列分散処理システム、１０２～１０５…コンピュータ

【図１】

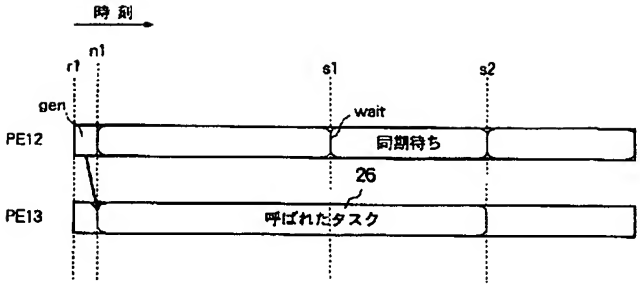


【図２】

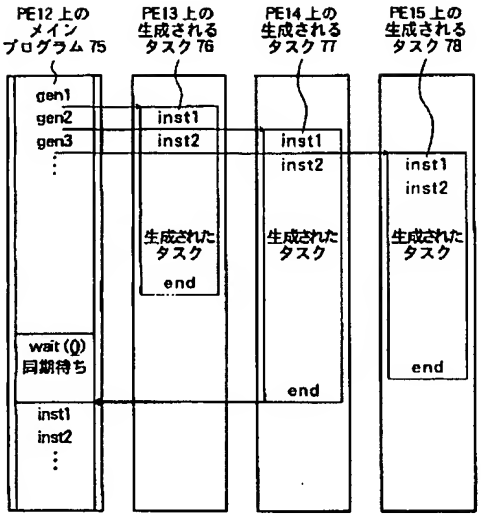


【図１０】

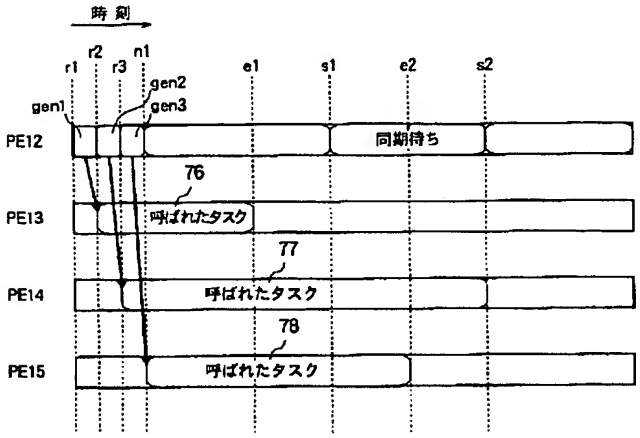
【図３】


時刻
r1
n1
s1
s2
gen
wait
PE12
同期待ち
26
PE13
呼ばれたタスク

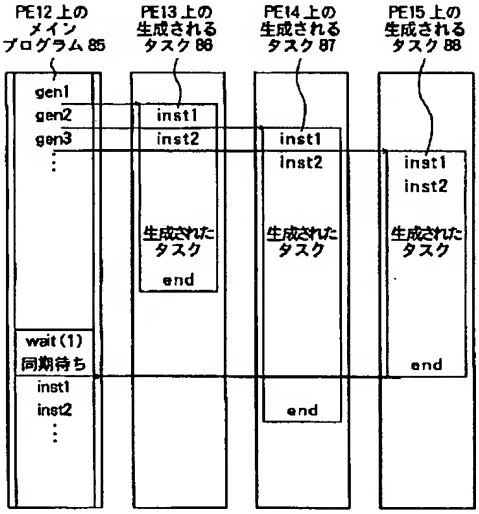
【図４】


PE12上の
メイン
プログラム 75
PE13上の
生成される
タスク 76
PE14上の
生成される
タスク 77
PE15上の
生成される
タスク 78
gen1
gen2
gen3
inst1
inst2
生成された
タスク
end
wait (0)
同期待ち

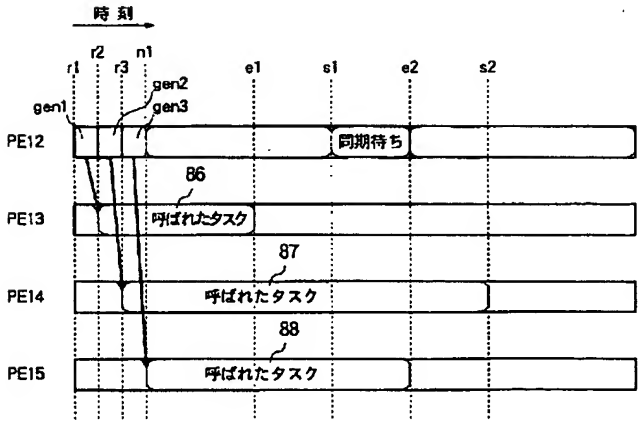
【図５】


時刻
r1
r2
r3
n1
e1
s1
e2
s2
gen1
gen2
gen3
PE12
同期待ち
76
PE13
呼ばれたタスク
77
PE14
呼ばれたタスク
78
PE15
呼ばれたタスク

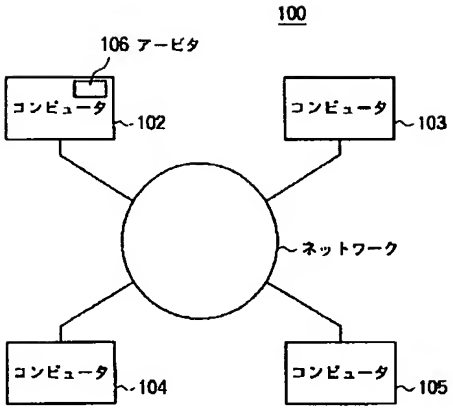
【図６】


PE12上の
メイン
プログラム 85
PE13上の
生成される
タスク 86
PE14上の
生成される
タスク 87
PE15上の
生成される
タスク 88
gen1
gen2
gen3
inst1
inst2
生成された
タスク
end
wait (1)
同期待ち

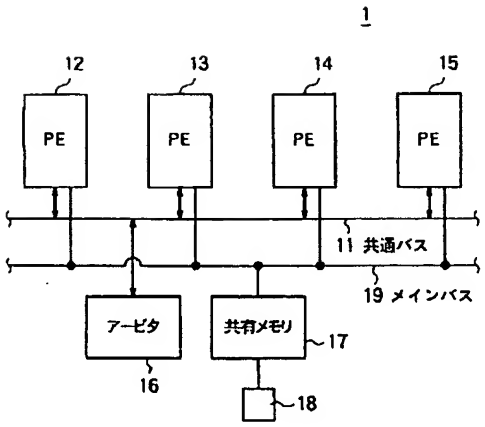
【図７】


時刻
r1
r2
r3
n1
e1
s1
e2
s2
gen1
gen2
gen3
PE12
同期待ち
86
PE13
呼ばれたタスク
87
PE14
呼ばれたタスク
88
PE15
呼ばれたタスク

【図８】


100
106 アービタ
コンピュータ
102
コンピュータ
103
ネットワーク
コンピュータ
104
コンピュータ
105

【図９】


1
12
13
14
15
PE
PE
PE
PE
11 共通バス
19 メインバス
アービタ
16
共有メモリ
17
18